VOLUME 2 ISSUE 3

NOVEMBER 1985

# **SUPER 99 MONTHLY**

| FOR | TH. |   |   |   |   |   |   |  |  |  |  |   |  |  | 1  |
|-----|-----|---|---|---|---|---|---|--|--|--|--|---|--|--|----|
|     | EME |   |   |   |   |   |   |  |  |  |  |   |  |  |    |
| 99  | POT | P | 0 | U | R | R | 1 |  |  |  |  | i |  |  | 11 |

Millers Graphics has announced the upcoming release of a new software package, DisKassembler<sup>TM</sup>. Written by Tom Freeman, DisKassembler<sup>TM</sup> creates directly assemblable source files from 99/4A Assembly Language object code that is in either Display Fixed 80 or memory image format (such as game files). In addition, it will disassemble console memory and all valid DSR's. Program output is to disk or any printer. Object files may be from floppy disk, hard disk or RAM disk in CorComp, MYARC or TI disk controller formats. The program is for anyone interested in how programs were constructed and in learning new programming techniques. Carrying a suggested price of \$19.95 (plus shipping and handling), the package will include complete and useful documentation (the hallmark of all MG products).

Kracker<sup>TM</sup> have been reported to be difficult to find in some regions, MG now offers the chips at \$4.50 each, with C.O.D. (\$1.90) being available for U.S. customers (other countries, prepaid). Installation is provided only for orders initiated with the optional chips specified (total price \$184.95 plus shipping and handling).

### FORTH

The first shipments of GRAM Kracker<sup>TM</sup>, Millers Graphics' incredible new hardware device, will be released on December 16 and 17. Due to quality control procedures that ensure that all customers will receive the product without jumper modifications, the shipment dates are behind original projections, which has prompted Millers Graphics to provide UPS Blue Label shipping at no extra charge to ensure arrival by Christmas.

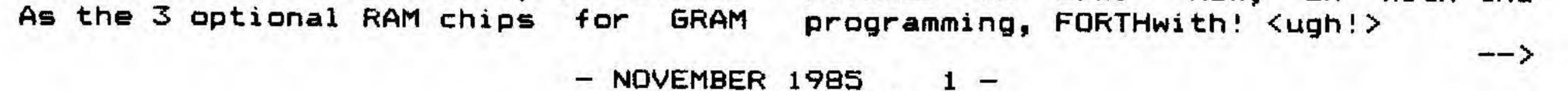
# Strings, Part 1

by Warren Agee

STANDARD: 1A 2EA 4B 5A 6B 7B 9B

# PREFACE:

With this tutorial (and more to come!), I humbly submit what I have learned by programming in the FORTH language. One reason I decided to put down into words the knowledge I have acquired is to share my experiences, frustrations and triumphs while hacking away with FORTH. But, on a more personal level, I give these tutorials to the TI world as a token of appreciation for everything I have gained from knowing such people as Ronald Albright, Barry Traver, and Howie Rosenberg, just to name a few, as well as the whole gang on the TI FORUM. These and many others have given unselfishly to both me and the TI community as a whole, and I am proud to be part of a community that refuses to die. Now, on with the



SUPER 99 MONTHLY

# STRINGING ALONG IN FORTH

Of all the peculiarities the beginner confronts in FORTH, string handling is a major obstacle. Nothing is more frustrating than to sit down and have no idea how to write something like A\$="1234"::A=VAL(A\$). No advanced string-handling routines come with the TI FORTH systems disk. So, it is up to the programmer to invent his own. Hopefully, this article will make it much easier to write a FORTH program that involves any string mainpulation at all.

# THE BASICS

Before jumping into the new string words, let's first take a look at how a string sits in memory. This knowledge is imperative in order to fully exploit the power of FORTH. Think of a string as a numeric array; each character in the string represents a number, or byte. The string HOME COMPUTER would look like this:

# IHIOIMIEL ICIOIMIPIUITIEIRI

The first "box" represents the address in memory where this string starts. Determining the location of this address is what we will discuss next.

There are many ways to store strings; we could save them in VDP RAM, or in the disk buffers. In this article, we will investigate storing strings directly in the dictionary. A string variable is no more than a numeric variable stretched out. In fact, unlike BASIC, there is only one type of variable in FORTH. The only thing that differs is the size. First use the word VARIABLE to create a variable. But when you create it, let's say 0 VARIABLE TEST, only two bytes are alloted for storage. This is fine for single numbers; but for strings, we can use ALLOT to specify the length of the variable. For instance, 0 VARIABLE TEST 8 ALLOT will create a variable with a length of ten bytes. This gives us room for a string with a maximum length of 10 characters. If the above is exectuted, the variable will look like this in memory:

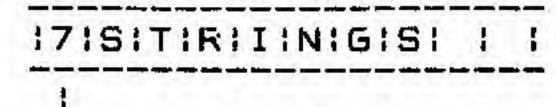
addr of TEST

Once the string is created in the dictionary, there may be garbage in the variable. Here we can use BLANKS to clean it out: TEST 10 BLANKS. This will fill ten bytes of memory, starting at TEST, with blanks (ASCII 32).

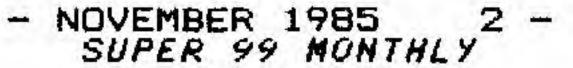
Now that space has been reserved for the string, there are basically two ways to store the string. If the contents of the variable is not going to change, then the word !" can be used. All this word requires is an address on the stack. So, to store STRINGS in the variable TEST defined above, the sequence TEXT !" STRINGS" will do the trick. If you wish the user to input the string, the word EXPECT is available, which is similar to BASIC's INPUT statement; it awaits an entry from the keyboard. EXPECT requires both an address and the maximum length of the string on the stack. Using TEST 7 EXPECT will achieve the same results as TEST !" STRINGS" . The variable will now look like this:

| 10  | 17  | - L  | D        | I T | A.I | 1 | C. | 1.6 | 21 | 1 | 1 | 1   |
|-----|-----|------|----------|-----|-----|---|----|-----|----|---|---|-----|
| 1 🖵 | 1.1 | - I. | <b>L</b> | 1 L | 14  | 4 | 0  | 1.1 |    |   | 1 | - H |

This presents our first problem. Since the contents of TEST is not expected to change, the length of the string can be assumed to always be 7. However, if the length will vary, we must keep track of it. EXPECT does not do this for us. Sure, it requires a length on the stack, but it does not incorporate this value into the string. Not to worry. This brings us to our first new word, ACCEPT, which replaces EXPECT. The only difference is that ACCEPT stores the actual length of the string entered into the byte preceding the string. This is often called the count byte. If we use ACCEPT in the example above, our string would now look like this:







As you can see, the first letter of the string, the "S", no longer sits at TEST; the whole string has moved over one byte to make room for the court. Now, to print this string is a trivial matter of using TEST COUNT TYPE. TEST supplies the addr of the complete string. COUNT takes that address, calculates the address of the actual string (TEST+1), and finally supplies the length of the string. Everything is ready for TYPE. To summarize what we have done so far, consider the following example:

> O VARIABLE COOKIE 18 ALLOT (reserves 20 bytes) COOKIE 20 BLANKS COOKIE 20 ACCEPT \_CHOCOLATE CHIP\_ COOKIE COUNT TYPE

Note: any words that appear between underscore characters (\_) are to be typed in as a response to the ACCEPT word.

# MOVING AROUND

Up till now, I have discussed performing basic functions on strings which reside directly in the dictionary. This is not always the ideal situation. A much better way is to store the string in a temporary spot, do what needs to be done, then move it back into the dictionary. This temporary spot is called PAD. Typing in PAD just leaves an address on the stack, just as TEST does. Typically, instead of typing in TEST 10 ACCEPT, you would type PAD 10 ACCEPT. Once any processing is done, the word CMOVE can move the bugger back to where it belongs. Here arises our second problem. CMOVE moves a specified quantity of bytes from low memory to high memory. But what if you want to go the other way around? Well, define a new word, of course! The new word will be <CMOVE, which is included in some versions of FORTH. But wait--isn't it rather a hassle having to remember which word to use? Of course it is! Remember, FORTH is extensible, and we can make it as user-friendly as we like! The next new word will be CMOVE\$, which decides which way the string is moving, and does the moving for you.

Here is an example of using CMOVE\$ and PAD:

```
O VARIABLE DRESSER 8 ALLOT
DRESSER 10 BLANKS
PAD 10 ACCEPT _SOCKS_
. (string processing done here)
PAD COUNT
                   (get addr and length)
1+ SWAP 1- SWAP
                      (PAD-1 CNT+1)
DRESSER SWAP
                      (PAD-1 DRESSER CNT+1)
CMOVE$
DRESSER COUNT TYPE
```

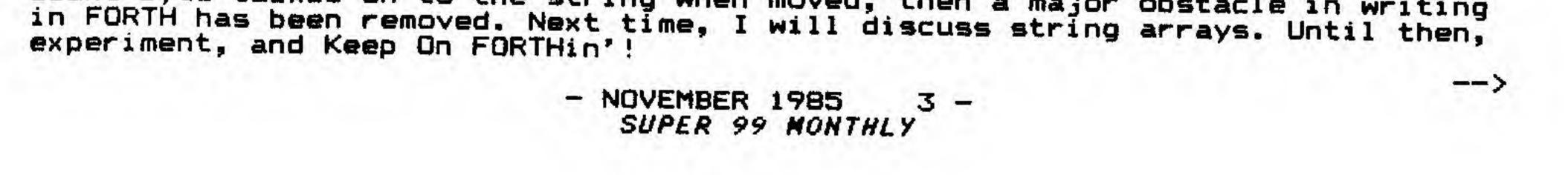
Everything should make sense until you get to the 1+ SWAP 1- SWAP. The reasoning is a little hard to grasp at first: we want to move SOCKS from PAD to DRESSER. We also want to maintain that ever-important count byte. But when we use PAD COUNT, we only have the addr and length of the string itself, not including the count. So we compensate. Add 1 to the count (because we want to move the count byte along with the string), then subtract one from the address. COUNT adds 1 to the address, so we have to correct this to catch the count. Once these two numbers have been corrected to catch the count byte, shift things around to get everything ready for CMOVE\$. To better illustrate this, here is a diagram of PAD:

1515101CIKIS1 1 1 1 1 (Contents of PAD)

| PAD+1 (This is where you are using PAD COUNT)

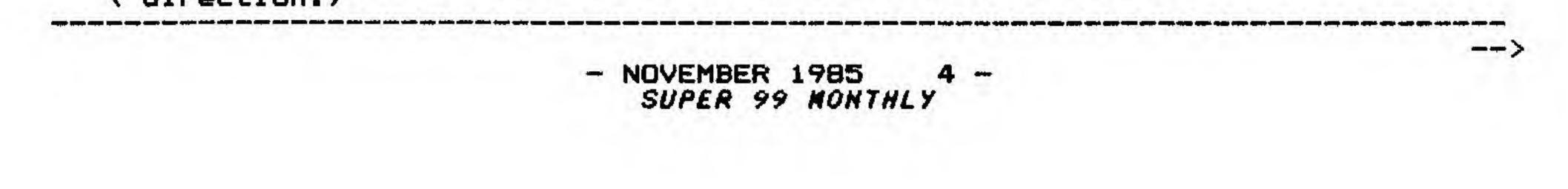
PAD (This is where you are using PAD COUNT 1+ SWAP 1- SWAP)

If you can understand the principle of the count byte, and how to keep the count byte tacked on to the string when moved, then a major obstacle in writing



SUMMARY OF RESIDENT WORDS -

```
Create a variable.
VARIABLE (n--)
                 Reserves n bytes in the dictionary.
ALLOT (n--)
BLANKS (addr n--) Fills n bytes with blanks.
EXPECT (addr n--) Waits for input; stores string at addr.
       (addr--) Returns addr and count of a string.
COUNT
      (adr1 adr2 n)Moves n bytes from adr1 to adr2, from low to
CMOVE
                     high memory.
PAD
                     Temporary storage place for strings.
         (---adr)
NEW WORDS
-----
: PICK ( n1 -- n2)
  2 * SP@ + @ ;
( Copies nith number to top of stack)
*****
: LEN
      (addr -- n)
  255 0 ( string max=255 characters)
  DO
    DUP I + C@
              ( looks for null)
    0 = IF
       I LEAVE ( I=length of string)
    ENDIF
  LOOP
  SWAP DROP :
( Returns the length of a string at addr.)
*****
: ACCEPT ( addr n -- )
  OVER 1+ DUP ROT ( adr+1 )
 EXPECT
 LEN
                   ( length of string)
  SWAP C! :
                   ( store count byte at addr )
( Waits for input; stores count at addr and string
  starting)
( at adr+1.)
*****
: <CMOVE ( adr1 adr2 n)
  DUP ROT + SWAP ROT
  1-DUP ROT +
  DO
   1- I CO OVER C! -1
  +L00P
  DROP :
( Moves n bytes from adr1 to adr2, from high to low memory.)
*****
: CMOVE$ (adr1 adr2 n)
OVER 4 PICK >
IF <CMOVE
ELSE CMOVE
ENDIF :
( Moves n bytes from adr1 to adr2; automatically decides on)
( direction.)
```

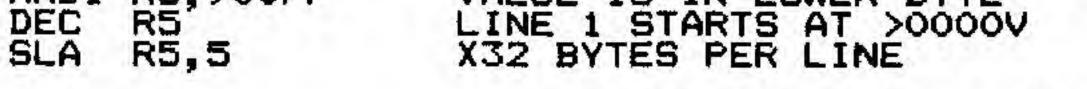


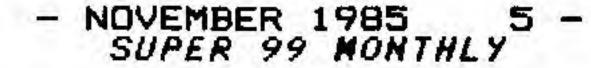
# ASSEMBLY

STANDARD: 1A 2XB EA TW 3B 4B 5A 6B 7B 9B 10B

TI-WRITER SCREEN DUMP inspired by May, 1985 Super 99 Monthly The following Source code, when assembled and combined with the XB calling routine and Subprogram will create a DISPLAY/VARIABLE 80 file that will print a screen image from the TI-WRITER FORMATTER. The program will work with any EPSON compatible printer. Insert the following line in your XB program where you want the dump to occur: CALL TIW\_DUMP(DE,F\$,BL,EL,T):: STOP DE= Density (1 or 2) Where F\$= Filename that you want the dump stored under For example: DSK1.PICTURE BL= Beginning line of the screen that you want saved EL= Ending line of the screen that you want saved T = Tab value Note: Tab of 20 centers picture Type in and save the following sub program in merged format. Merge it into the program that contains the graphics that you want dumped. 25000 SUB TIW\_DUMP (DE, F\$, BL, EL, T) 25010 ON ERROR 25080 25020 IF (T<0)+(T>40)+(BL>EL)+(BL<1)+(BL>24)+(EL<1)+(EL>24)THEN GOSUB 25080 25030 IF DE<>2 THEN DE\$="DE1" ELSE DE\$="DE2" 25040 CALL INIT :: CALL LOAD ("DSK1.TIWDUMP-D"):: CALL LINK (DE\$, F\$, BL, EL, T) 25045 ! LINES 25050 to 25070 MAY BE DELETED IF DESIRED

| *                 | 2505<br>2504<br>.PL<br>2507<br>2507<br>2507<br>2509 | 1 WILL STOP<br>70 CLOSE #1<br>75 SUBEXIT  | <pre>\$,DISPLAY,VARIABLE 80,APPEND<br/>CHR\$(27)&amp;CHR\$(64):".PL 1" ! 27-64 RESETS PRINTER,<br/>UNWANTED FORM FEED<br/>&gt; PARAMETER" :: STOP :: RETURN<br/>************************************</pre> |
|---|---|---|--|
| ¥<br>* by<br>*  | Josep   | h H. Spiege<br>E ID: T16240   | COMPUSERVE ID 72426,3432   |
| VSBW<br>VMBW<br>VSBR<br>VMBR<br>STRREF<br>NUMREF<br>FAC |   | DE1,DE2<br>>2020<br>>2024<br>>2028<br>>2028<br>>2028<br>>2020<br>>2014<br>>2000<br>>834A<br>>2700 | ***************************************  |
| DE 1  | MOV<br>LWPI<br>CLR<br>JMP                           | R11, @SAVE<br>MYREGS<br>R14<br>MAIN   | SAVE RETURN ADDRESS<br>RESET FLAG -> SINGLE DENSITY  |
| DE2   |   | R11,@SAVE<br>MYREGS   | SAVE RETURN ADDRESS<br>SET FLAG -> DOUBLE DENSITY  |
| ******<br>*<br>*******                                  | GET 5   | START AND EN  | **************************************   |
| MAIN<br>GLINE   | LI<br>LI<br>CLR<br>BLWP<br>MOV                      | R4, STARTL<br>R1, 2<br>R0<br>@NUMREF<br>@FAC, R5<br>R5, >00FF                                     | POINT TO LOCATION TO HOLD START ADDRESS<br>START LINE IS SECOND VALUE FROM XB<br>GET VALUE PASSED FROM XB<br>MOVE VALUE FROM FAC TO R5<br>VALUE IS IN LOWER BYTE   |





-->

| CI<br>JLT<br>CLR<br>BLWP<br>CLR<br>MODI<br>AND<br>AND<br>AND<br>AND<br>AND<br>AND<br>AND<br>AND<br>AND<br>AND | R1<br>R1,4<br>GLINE<br>R0<br>@NUMREF<br>R5<br>@FAC,R4<br>R5<br>@FAC,R4<br>R4,>00FF<br>R5<br>R4,-10<br>C3<br>L00P3<br>R5<br>R5<br>R4,10 |
|---|--|
| CLR<br>LII<br>BLWP<br>LII<br>BLWP<br>DATA<br>LII<br>BLWP  |  |

SAVE VALUE FOR LATER END LINE STORED AFTER START LINE GET READY TO GET NEXT VALUE FROM XB BOTH START AND END LINE STORED? NO, GET END LINE YES, GET TAB VALUE MOVE VALUE FROM FAC TO R4 VALUE IS IN LOWER BYTE START BINARY TO BCD CONVERSION LOOP3 **R5 COUNTS "TENS"** R4 COUNTS "ONES" **C**3 STORE "TENS" AS HIGH BYTE OF "ONES" CONVERT TO ASCII STORE IN TAB PORTION OF FIRST TL. \* NOW WE WANT THE FIRST VALUE FROM XB STORE IT AS PART OF THE PAB GET THE STRING NOW VDP BUFFER FOR PAB MOVE IT TO VDP FROM CPU NOW OPEN THE DISK FILE 1 MOVE WRITE BYTE TO PAB \* SINGLE DENSITY DUMP? MOV R14, R14 YES, DON'T CHANGE ANYTHING JEQ SD NO, CHANGE DENSITY AND INC **@DENS** PRINT LINE LENGTH IN FIRST TL. INC **@LEN** RO, >1E05 R1, >2B00 SD LI LENGTH OF FIRST TL LI MOVE IT TO PAB BLWP @VSBW FIRST TL CONTAINS CODES TO INITIALIZE GRAPHICS RO, >1F00 DATA BUFFER IN VDP LI R1, TL1 R2,>2B LI MOVE FIRST TL TO VDP BLWP QVMBW MOV R6,@>8356 SEND IT TO THE PRINTER BLWP @DSRLNK DATA 8 EACH REDEFINABLE XB CHARACTERS PATTERN WILL BE STORED AS A TRANSLITERATE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* FOINT TO START OF IMAGE TABLE R10,1024 LI R10, R0 LO MOV WE'LL STORE THE PATTERN HERE R1, IN LI R2,8 LI GET A PATTERN BLWP **@VMBR** R5 POINTS TO BIT BEING CONVERTED R5,128 LI R8 POINTS TO BYTE IN CONVERTED PATTERN R9 POINTS TO BYTE NUMBER R3 POINTS TO BYTE BEING CONVERTED R8 CLR L3 R9,128 LI CLR R3 HOLDS CONVERTED BYTE R4 CLR R4 R7 HOLDS BYTE BEING CONVERTED L2 CLR **R7** \*\*\*\* CONVERT PATTERN \*\*\*\*\*\*\*\*\*\*\*\*\* MOVB @IN(3),R7 SWPB R7 C



| L1    | S RS,R7<br>SWPB R7<br>MOVB R7,@IN(3)<br>INC R3<br>SRA R9,1<br>JGT L2<br>SWPB R4<br>MOVB R4,@D0(8) |
|-------|---|
| ***** | INC R8<br>SRA R5,1<br>CI R8,8<br>JLT L3   |
| *     | CHANGE TO ASCII VALUES AND STORE IN OUTPUT BUFFER *   |
| ***** | ******************  |
|       | CLR R9 POINTS TO BYTE IN CONVERTED PATTERN<br>CLR R8 OFFSET FOR OUTPUT BUFFER                     |
| LDTL  | ANOTHER BINARY TO BCD CONVERSION *<br>CLR R4 R4 R4 COUNTS "ONES"                                  |
|       | CLR R5 R5 COUNTS "TENS"   |
|       | CLR R7 R7 COUNTS "HUNDREDS"<br>MOVE @DO(9),R4   |
|       | SWPB R4   |
| LOOP  | INC R5  |
|       | AI R4,-10<br>JLT C1   |
|       | JMP LOOP  |
| C1    | DEC R5  |
|       | AI R4,10<br>CI R5,10  |
|       | JLT L100  |
| LOOP2 | INC R7  |
|       | AI R5,-10<br>JLT C2   |
| 122   | JMP LOOP2   |
| C2    | DEC R7  |

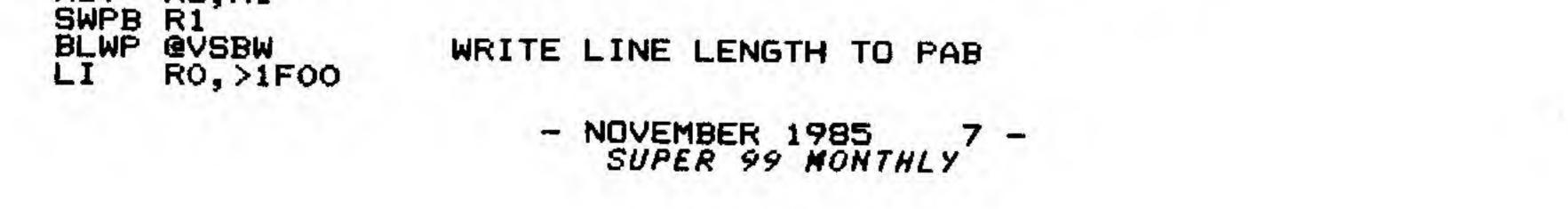
.

-->

1

3

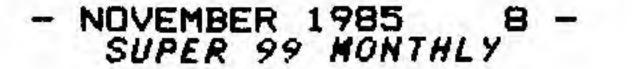
R5,10 AI DON'T PRINT ANY LEADING ZEROS HERE \* MOV JEQ MOVB L100 R7,R7 ZERO1 @ASCII(7),@TLDATA(8) R8 R5,R5 INC MOV R5,R5 JEQ ZERO2 MOVB @ASCII(5),@TLDATA(8) INC R8 MOVB @ASCII(4),@TLDATA(8) INC R8 MOVB @ASCII(4),@TLDATA(8) INC R8 ZER01 ZERO2 INC R8 MOVB @COMMA,@TLDATA(8) INC MOV JEQ JEQ **R8** R14, R14 SINGLE DENSITY? SD6 R7,R7 ZERO3 IF NOT, REPEAT LAST CHARACTER IN BUFFER @ASCII(7),@TLDATA(8) R8 R5,R5 MOVB INC ZERO3 MOV JEQ ZERO4 MOVB @ASCII(5),@TLDATA(8) INC R8 MOVB @ASCII(4),@TLDATA(8) INC R8 R8 ZERO4 MOVB @COMMA, @TLDATA(8) INC **R8** SD6 INC **R**9 C1 R9,8 LAST BYTE? LDTL JLT IF NOT, GET NEXT OUTPUT TRANSLITERATE \*\*\*\*\*\*\*\*\*\* \*\*\*\*\* COMPUTE TOTAL LINE LENGTH GET NEXT ASCII TRANSLITERATE VALUE R8.7 @NXT AI BL LI RO, >1E05 MOV R8, R1



|                | MOV                     | R1,TLBUF<br>R8,R2                |  |
|----------------|-------------------------|----------------------------------|--|
| SD4            | BLWP                    | evmbw                            | PUT LINE IN VDP  |
|                | BLWP                    | R6,@>8356<br>@DSRLNK             | NOW OUTPUT IT TO DISK  |
|                | DATA<br>AI<br>CI<br>JGT | 8<br>R10,8<br>R10,1903           | POINT TO NEXT IMAGE<br>LAST ONE?   |
|                | B                       | SCDMP<br>@LO                     | IF NOT, DO NEXT ONE  |
| *****          | *****<br>DUMP           | ***********                      | ***************************************                                      |
| *****          | *****                   | IMAGE TO DIS                     | a ta                                     |
| SCDMP          | ĻΙ                      | R0, >1E05                        |  |
|                | BLWP                    | R1,>2100<br>@VSBW                | PUT LENGTH OF IMAGE LINE IN PAB  |
|                | MOV                     | ESTARTL, R5                      | GET STARTING LOCATION AND<br>ENDING LOCATION                                 |
|                | MOV                     | GENDL,R7<br>R7                   | ENDING LUCHTION  |
| LOOPB          | CLR                     | R4                               |  |
| LOOPC          | BLWP                    | R5, RO<br>@VSBR                  | READ CHARACTER FROM IMAGE TABLE  |
|                | SRL                     | R1,8                             | MOVE TO LOWER ORDER BYTE   |
|                | AI                      | R1, -96<br>R1, 32                | ADJUST FOR BASIC<br>LESS THAN LEGAL GRAPHIC CHAR?                            |
|                | AI<br>CI<br>JGT         | CONT1                            | LEGG INAN LEGAL ONAFRIC COANT  |
| CONTA          | LI_                     | R1,32<br>R1,143<br>CONT2         | IF SO, DEFAULT TO CHR\$(32)  |
| CONT1          | JLT                     | CONT2                            | GREATÉR THAN LEGAL?  |
| and the second | LI<br>AI                | R1,143                           | IF SO, DEFAULT TO CHR\$(143)<br>ADJUST R1 TO BECOME OFFSET FOR "SCREEN" DATA |
| CONT2          | AI<br>MOVB              | R1,143<br>R1,-32<br>@SCREEN(1),( | ADJUST R1 TO BECOME OFFSET FOR "SCREEN" DATA                                 |
|                | INC                     | R4                               | SDUF DIHAH/  |
|                | INC                     | R5                               |  |
|                | CI<br>JLT               | R4,32<br>LOOPC                   | END OF LINE?<br>IF NOT, GET NEXT IMAGE                                       |
|                | LI                      | RO, >1F00                        |  |
|                | I The                   | R1_BUFFFR                        |  |

RI, DUFFER See all INC **R4** MOV R4, R2 BLWP QVMBW IF SO, MOVE LINE TO VDP MOV R6, 8>8356 BLWP @DSRLNK THEN OUTPUT TO DISK DATA 8 R5, R7 LAST LINE? C IF NOT, DO NEXT JLT LOOPB RESET TRANSLITERATE CODES \*\*\*\*\*\*\* R0,>1E05 LI R1, >0BOO @VSBW LI CHANGE LINE LENGTH IN PAB BLWP R4,>3030 LI MOV R4, @DEC3 AI R4, >0100 / MOVB R4, @DEC1 / MOVB @DEC3, @TLDATA MOVB @DEC2, @TLDATA+1 MOVB @DEC2, @TLDATA+1 MOVB @DEC1, @TLDATA+2 LI R0, >1F00 RESET TRANSLITERATE BUFFER TO .TL 001 RST TRANSLITERATE THE VALUE TO ITSELF R1, TLBUF R2, >000B GVMBW PUT IT IN VDP BLWP MOV R6, @>8356 @DSRLNK OUTPUT IT TO THE DISK DATA 8 MOV @DEC3,R5 R5,>3132 L12 CI JLT. HAV" ALL VALUES MOVB @DEC1,R5 **BEEN RESET?** R5,8 R5,>32 EXIT SRL CI JEQ IF YES, GET READY TO RETURN L12 BL **ONXT** NOT, CALCULATE NEXT TL VALUE IF JMP RST \*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*

### 



-->

| EXIT                  | LI RO, >1E00<br>LI R1, >0100<br>BLWP RVSBW    | DUT CLOSE DVTC IN DAD                                       |
|-----------------------|---|---|
|                       | MOV R6, 0>8356<br>BLWP @DSRLNK                | PUT CLOSE BYTE IN PAB<br>CLOSE FILE                         |
|                       | DATA 8<br>LWPI >83E0<br>MOV @SAVE.R11         | RESET WS POINTER<br>GET RETURN VALUE<br>RETURN TO XB        |
| *****                 | B #R11<br>*********                           | ***************************************                     |
| *****                 | ROUTINE TO INCRE                              | MENT ASCII TL VALUE<br>************************************ |
| NXT                   | CLR R4<br>MOVB @DEC1,R4                       | MOVE "ONES" BYTE TO R4                                      |
|                       | AI R4, >0100<br>MOVB R4, @DEC1                | INCREMENT IT AND MOVE<br>IT BACK                            |
|                       | CI R4,>3A00<br>JLT L10                        | IS IT GREATER THAN ASCII 9 (CHR\$(57))?                     |
|                       | LI R4,>3000<br>MOVB R4,@DEC1<br>MOVB @DEC2,R4 | IF SD, REPLACE THE VALUE WITH ASCII O<br>AND INCREMENT      |
|                       | AI R4, >0100                                  | THE "TENS"  |
|                       | CI R4,>3A00                                   | IS THE "TENS" VALUE GREATER THAN ASCII 97                   |
|                       | JLT L10<br>LI R4,>3000                        |   |
|                       | MOVB R4, @DEC2<br>MOVB @DEC3, R4              | IF SO, REPLACE THE VALUE WITH ASCII O<br>AND INCREMENT THE  |
| **                    | AI R4,>0100<br>MOVB R4, QDEC3                 | "HUNDREDS"<br>VALUE   |
| **<br>**<br>**<br>L10 | CHECK IF THE VAL<br>WANT TO TRANSLIT          | UE IS ONE THAT WE DON'T                                     |
| IIO                   | MOVB @DEC1, R9                                |   |
|                       | SWPB R9                                       |   |
|                       | MOVB @DEC2,R9<br>CI R9,>3130<br>JEQ NXT       |   |
|                       | CI R9,>3133<br>JEQ NXT                        |   |
|                       | CI R9,>3237<br>JEQ NXT                        |   |
|                       | CI R9,>3332                                   |   |
|                       | CI R9,>3338                                   |   |
|                       | JEQ NXT<br>CI R9,>3432                        |   |
|                       | JEQ NXT<br>CI R9,>3436                        |   |
|                       | JEQ NXT<br>CI R9,>3634                        |   |
|                       | JEQ NXT<br>CI R9,>3934                        |   |
|                       | JEQ NXT<br>RT                                 | RETURN WHEN OK  |
| *****                 | *******                                       | ***************************************                     |
| * NOTE                |   | NDED BASIC LOADER DOES NOT RECOGNIZE THE DSRLNK *           |
| *                     |   | *   |
| * BEG                 | INNING OF DSRLNK R                            |   |
| #<br>DSRLNH           |   |   |
| DSRO                  | MOV \$14+,5<br>SZCB @DATA2,15                 |   |
|                       | MOV @>8356,0<br>MOV 0,9                       |   |
|                       | AI 9 >FFF8<br>BLWP QVSBR                      |   |
|                       | MOVB 1,3<br>SRL 3,8                           |   |
|                       | SETO 4  |   |
| DBP2                  | TNC O   |   |

0.1

5

-

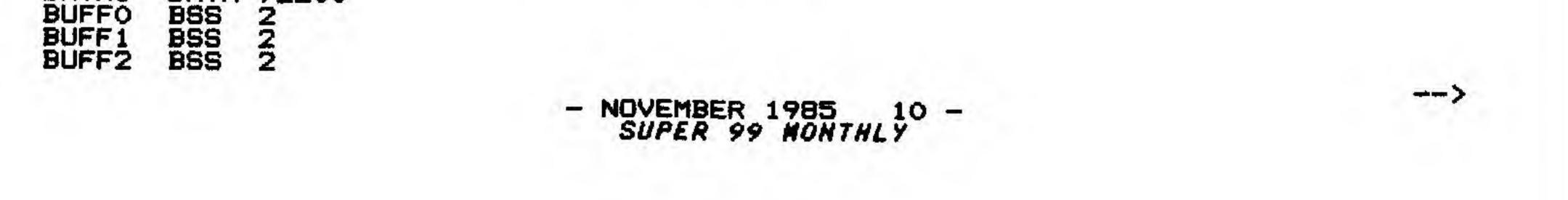
,



| NAME<br>DSRREG<br>DATA1<br>DATA2<br>DATA3<br>BUFFO | BSS<br>BSS<br>DATA<br>DATA<br>DATA<br>BSS                    | 14<br>32<br>>AAOO<br>>2000<br>>2EOO   | NAME BUFFER<br>WORKSPACE FOR | DSRLNK |
|--|--|---|------------------------------|--------|
| DSR5<br>DSR3<br>DSR11                              | RTWP<br>LWPI<br>CLR<br>SWPB<br>MOVB<br>SOCB<br>RTWP          | DSRREG<br>1<br>1,*13<br>@DATA2,15   |                              |        |
| DSR8   | INC<br>MOV<br>MOV<br>BL<br>JMPZ<br>JMPZ<br>JNE<br>SRL<br>JNE | 1<br>1,@BUFF5<br>9,@BUFF2<br>12,@BUFF1<br>#9<br>DSR9<br>0<br>DSR89<br>0<br>DSR86<br>9,0<br>@VSBR<br>1,13<br>DSR11 |                              |        |
| DSR10  | SRL<br>LI<br>CB<br>JNE<br>DEC<br>JNE                         | 5,8<br>6,NAME<br>*6+,*2+<br>DSR9<br>5<br>DSR10  | ÷                            |        |
| DOK/   | JEQ<br>MOV<br>INCT<br>MOVB<br>JEQ<br>CB<br>JNE               | #2.2<br>DSR6<br>2,@>83D2<br>2<br>#2+,9<br>@>8355,5<br>DSR8<br>5,#2+<br>DSR9                                       |                              |        |
| DSR7   | SBO  | @>83D2,2  |                              |        |
| DSR9   | MOV<br>SBO<br>LIB<br>JNE<br>JNE<br>JMP<br>MOV                | 12,@>83D0<br>0<br>2,>4000<br>*R2,@DATA1<br>DSR6<br>@DSRREG+10,2<br>DSR7<br>@>07D2 7                               |                              |        |
| DSR4   | AI<br>CLR<br>CI<br>JEQ                                       | 12,>0100<br>@>83D0<br>12,>2000<br>DSR5  |                              |        |
| DSR6   | MOV<br>LWPI<br>CLR<br>LI<br>MOV<br>JEQ<br>SBZ                | @\$8356,@BUFF<br>>83E0<br>1<br>12, >0F00<br>12, 12<br>DSR4<br>0   | 4                            |        |
|  | CI<br>JGT<br>CLR<br>MOV<br>MOV<br>INC                        | 4,7<br>DSR3<br>@>83D0<br>4,@>8354<br>4,@BUFF3<br>4<br>4,@>8356  |                              |        |
| DSR1   | CB<br>JNE<br>MOV<br>JEQ                                      | 1, CDATA3<br>DSR2<br>4,4<br>DSR3  |                              |        |
|  | C<br>JEQ<br>BLWP<br>MOVB                                     | 4.3<br>DSR1<br>@VSBR<br>1.*2+   |                              |        |

.

.



.....

-**BUFF4** BSS **BUFF5** 2 855 END OF DSRLNK ROUTINE MYREGS BSS 32 SAVE DATA >0000 ASCII DATA >3031, >3233, >3435, >3637, >3839 COMMA DATA >2000 PAB DATA >0012,>1F00,>5000,>0000 BYTE >00 FILE BYTE >1F BSS >1F EVEN TL1 TEXT 1:27,65,8,10,13,27,68,' '.TL TAB '18' TEXT ,0,9,27,' '75' TEXT DENS TEXT TEXT **9**, ' 2 LEN TEXT \* 1 BYTE CR >OD EVEN BSS IN 8 BSS DO 8 TEXT TLBUF '.TL ' DEC3 BYTE >30 DEC2 BYTE >30 DEC1 BYTE >31 BYTE >3A TLDATA BSS 72 EVEN BUFFER BYTE >01 BUFDTA BSS 32 EVEN >0000 STARTL DATA ENDL DATA >0300 SCREEN DATA >0203, >0405, >0607, >0809 >OBOC, >OEOF, >1011, >1213, >1415 DATA DATA

THIS IS A TABLE OF

| DHIH | /101/,/1017,/1416,/101E,/1F21     |
|------|-----------------------------------|
| DATA | >2223, >2425, >2728, >2928, >2C2D |
| DATA | >2F30, >3132, >3334, >3536, >3738 |
| DATA | >393A, >383C, >3D3E, >3F41, >4243 |
| DATA | >4445, >4647, >4849, >4A48, >4C4D |
| DATA | >4E4F, >5051, >5253, >5455, >5657 |
| DATA | >5859, >5A5B, >5C5D, >5F60, >6162 |
| DATA | >6364, >6566, >6768, >696A, >686C |
| DATA | >6D6E, >6F70, >7172, >7374, >7576 |
| DATA | >7778, >797A                      |
| END  |                                   |

ALL THE CHARACTERS

(IN HEX) THAT WE WILL

TRANSLITERATE

# 99 POTPOURRI

News, Corrections, Updates, Editorials, Kudos and Come-what-may

I WISH I HAD:

44 146 7 7 14

۲.

1

5

Hicksville, NY.

Fulfillments:

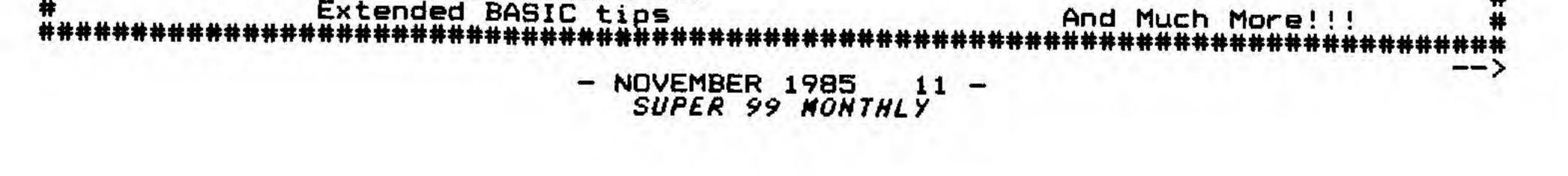
F2: For John Singleton, Westlake, LA. MENGEN, available on the TI FORUM on CIS, converts an Extended BASIC screen to Assembly object code for linking to your program. Graphics are supported, except character 130. A few screens can be loaded at once and using CALL INIT will allow loading another set of screens (your RAM Disk will help!).

### Wishesı

W3: A program to dump graphics and text to my Pro-Writer #8510 printer. I'd like to press a <CTRL> or <FCTN> key for the dump. F.J. Bubenik, Jr., The former manager of NCC has now formed her own discount disk firm. Contact Renee' Dezarn, 87 Rhoades Court, San Jose, CA 95126 today!

COMING SOON:

Surprises! New products from Bytemaster and more new staff members for Super 99 Monthly!



SUPER 99 NONTHLY ORDER FORM SUBSCRIPTIONS (PER YEAR): NAME\_\_\_\_\_ U.S. AND POSSESSIONS FIRST CLASS \$16.00 THIRD CLASS \$12.00 ADDRESS OTHER COUNTRIES \$26.50 CITY\_\_\_\_STATE\_\_\_\_ AIR MAIL SURFACE MAIL \$16.00 ZIP\_\_\_\_COUNTRY\_\_\_\_ INDIVIDUAL COPIES: U.S. SUBSCRIBERS FIRST CLASS \$ 1.35 THIRD CLASS \$ 1.00 CANADA SUBSCRIBERS \$ 1.35 For back issues, specify which: OTHER \$ 1.50 Check or Money Order in U.S. funds, READER FEEDBACK: (Attach comments) coded for processing through the U.S. Federal Reserve Bank System. No billings or credit sales. (all issues available at press time) STANDARD KEY | SUPER 99 MONTHLY is published monthly | by Bytemaster Computer Services, 171 | Mustang Street, Sulphur, LA 70663. 1 1 Computer A TI-99/4A 1 2 Module XB Extended BASIC All correspondence received will be TW TI-Writer considered unconditionally assigned subject to editing and comments by 1 3 RS-232 B TI the editors of SUPER 99 NUNTHLY Each contribution to this issue and 15 Expansion Box A TI the issue as a whole Copyright 1985 | 6 Disk B CorComp

| rights reserved. Copying done for<br>other than personal archival or<br>internal reference use without the<br>permission of Bytemaster Computer<br>Services is prohibited. Bytemaster<br>Computer Services assumes no<br>liability for errors in articles. | 7 Memory Card B MYARC MEXP-1<br>(128K)<br>9 Monitor or TV B TI Color Monitor<br>10 Printer B Gemini 15-X<br>GRAM Kracker and DisKassembler are<br>registered trademarks of Millers<br>Graphics. |
|--|---|
| EDITOR   | CORRESPONDING STAFF WRITERS   |
| Richard M. Mitchell (CIS 70337,1011)   | Barry A. Traver<br>Charles M. Robertson<br>Steven J. Szymkiewicz, MD  |
| Bytemaster Computer Services<br>171 Mustang Street<br>Sulphur, LA 70663  | Bulk Rate<br>  U.S. Postage<br>  PAID<br>  Sulphur, LA 70663<br>  Permit No. 141  |
| POSTMASTER: ADDRESS CORRECTION REQUE   | STED.   |
| RUSH TIME DATED MATER  |   |

